Official

7/7/03

REMARKS

The Office Action of April 8, 2003, has been carefully considered. Claims 1 - 48 are pending in the application. Claim 2 was objected to because of informalities. Claims 1-13, 45, and 48 were rejected under 35 U.S.C. § 112, second paragraph, as being indefinite. Claims 19, 20, 22-26, 33, 35, 37, and 43-48 were rejected under 35 U.S.C. § 102(b) as being anticipated by U.S. Patent No. 5,815,720 to Buzbee (hereinafter Buzbee). Claims 1-11, 13-15, 17, 18, 28-32, 38, 41, and 42 were rejected under 35 U.S.C. § 103(a) as being unpatentable over U.S. Patent No. 5,950,003 to Kaneshiro et al. (hereinafter Kaneshiro) in view of "Compilers: Principles, Techniques, and Tools" by Alfred Aho (hereinafter Aho). Claim 12 was rejected under 35 U.S.C. § 103(a) as being unpatentable over Kaneshiro in view of Aho and further in view of U.S. Patent No. 5,815,714 to Shridhar (hereinafter Shridhar). Claims 21, 27, 34, and 36 were rejected under 35 U.S.C. § 103(a) as being unpatentable over Buzbee in view of Kaneshiro. Claims 39 and 40 were rejected under 35 U.S.C. § 103(a) as being unpatentable over Aho in view of Kaneshiro.

The following REMARKS section is divided into the following sub-sections: Objections, Rejections, and Conclusion. The Rejection sub-section is further divided into § 112 rejections, § 102 rejections, and § 103 rejections. In overview, by the present amendment, Claims 1, 2, 10, 28, 34, 36, 45, and 48 have been amended, new claims 49-56 have been added, and the rejections have been traversed in view of the following remarks that include a brief description of certain aspects of the invention and a brief description of the teachings in the cited art.

I. OBJECTIONS

Claim 2 was objected to because of informalities, specifically that "an annotation information" should be "annotation information". The Applicant has reviewed the Examiner's recommendation. Claim 1, from which Claim 2 depends, has been amended to recite "generating annotation information" rather than "a non-executable statement." In light of this amendment to Claim 1, Applicant has applied the Examiner's recommendation for Claim 2 to amended Claim 1 so that the terminology is grammatically correct.

Official

7/7/03

II. REJECTIONS

A. 35 U.S.C. § 112 Claim Rejections

Claims 1-13, 45, and 48 were rejected under 35 U.S.C. § 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which the Applicant regards as the invention.

Claims 1-13 : The Examiner stated that there was insufficient antecedent basis for "the annotation information" recited in Claim 1. In addition, the Examiner stated that "a non-executable statement" recited in Claim 1 was unclear because the "Microsoft Computer Dictionary: Third Edition" defines a "statement" as "the smallest executable entity within a programming language." Therefore, the Examiner stated that "a non-executable statement" is interpreted to mean "information" or "data".

The Applicant acknowledges that the Microsoft Computer Dictionary defines a "statement" as "the smallest executable entity within a programming language". However, the Applicant contends that dictionary definitions are not determinative in isolation. Rather, when the general meaning of the term is gleaned from reference sources, such as dictionaries, the general meaning must always be compared against the use of the term in context. *Brookhill-Wilk 1 LLC v. Intuitive Surgical Inc.*, 66 USPQ.2d 1517, 1521 (CA FC 2003). For the present application, the term "non-executable statement" appears on page 13, line 1 of the specification. As stated on the same line, an example of a non-executable statement is "annotation debug information." Elsewhere in the specification, the term "annotation information" is used to broadly describe what is generated from the intermediate language. For example, the flowchart in FIGURE 3 illustrates emitting annotation information from the intermediate language code at block 330. Thus, the Applicant has amended Claim 1 to recite the more general term "annotation information" that includes annotation debug information, non-executable statements, and the like. Thus, with this amendment of Claim 1, the Applicant submits that the rejections of Claims 1-13 based on 35 U.S.C. § 112 have been overcome.

Claim 10 : The Examiner stated that the term "a second function" was unclear because the claim or parent claims did not refer to "a first function". The Applicant has reviewed the Examiner's recommendation and has amended Claim 10 to recite "a function". Thus, with this amendment of Claim 10, the Applicant submits that the rejection of Claim 10 based on 35 U.S.C. § 112 has been overcome.

Claim 13 : The Examiner stated that there was insufficient antecedent bases for the limitation "the arguments". The Applicant has amended Claim 13 to overcome the lack of antecedent bases for "the arguments". Additional claims, Claims 49 and 50, were added to claim aspects related to arguments. Thus, with this amendment of Claim 13, the Applicant submits that the rejection of Claim 13 based on 35 U.S.C. § 112 has been overcome.

Claims 45 and 48 : The Examiner stated that the term "the computer program analysis tool further comprises a computer program analysis tool" was unclear. Specifically, the Examiner questioned whether the term meant that a computer program analysis tool comprises within itself *another* computer program analysis tool. The Applicant draws the Examiner's attention to page 17, lines 3-4 of the specification, where the specification states that the computer program analysis tool may be one of several different types of tools, such as a profiler, a fault injector, or an optimizer. In one embodiment, an output of the computer program analysis tool may be read as input to a second computer program analysis tool. Thus, the present invention does not require compiling a special version of the executable program for each analysis tool because each analysis tool can use the annotation information that existed in the source before compilation of the source. Page 17, lines 5-7. The Applicant has amended Claims 45 and 48 to reflect this aspect of the invention. Thus, with the amendments to Claims 45 and 48, the Applicant submits that the rejection of Claims 45 and 48 based on 35 U.S.C. § 112 has been overcome.

B. 35 U.S.C. § 102 Claim Rejections

Claims 19, 20, 22-26, 33, 35, 37, and 43-48 were rejected under 35 U.S.C. § 102(b) as being anticipated by Buzbee. In review, in order for prior art to anticipate a claim under 35 U.S.C. § 102 every element of the claimed invention must be identically disclosed either

expressly or under principles of inherency in a single reference. Further, the exclusion of a claimed element from a prior art reference, no matter how insubstantial, is enough to negate anticipation by that reference. The test of whether anticipation exists in a particular case is a question of fact, and is applied element-by-element to a single prior art reference. Only if the prior art literally reads on every element of the rejected claim will the claimed invention be anticipated under this test.

The present invention is directed at analysis tools and methods that read annotation information generated from annotation representations within source code. The analysis tools include debuggers, instruction counters, profilers, fault-injectors, resource failure simulators, optimizers, and the like. Page 1, lines 11-12. The tools and methods are adapted to modify executable code generated from source code based on annotation information. For example, if the analysis tool is a debugger, the debugger may insert debugging code into the executable code. Other tools may be customized to interpret the annotation information in a manner to reflect their desired goal. Thus, several different tools may use the same executable code that has been generated from one source code. The annotation representation within the source code may be an intrinsic function. The annotation information used by the tools remains associated with the code associated with the annotation representation (e.g., the intrinsic function). Page 5, line 16 to page 6, line 11.

Independent Claim 19 and its dependent Claims 20, 22-23

Buzbee is directed at profilers that gather profile information. The profile information is then later used to optimize object code of an application. The invention is directed at reducing the number of compilations necessary for optimizing object code. Thus, Buzbee only requires compiling the application source code two times (Col. 3, lines 58-59), rather than requiring compilation of the application source code three times as needed in prior systems (Col. 1, lines 54-63).

Buzbee teaches to first compile the source to produce a first object code for the application (Col. 2, lines 3-4). The first object code is dynamically translated to produce a

second object. The second object code includes profiling code which, when executed, produces the profile information. The source code for the application is then recompiled using the profile information to produce an optimized object code (Col. 2, lines 6-15).

Rejected independent Claim 19, from which rejected Claims 20 and 22-23 depend, recites "reading annotation information in an executable computer program". This is quite different than what is taught in Buzbee. As described above, in Buzbee, source code is compiled to produce a first object code. The first object code is dynamically translated to produce a second object that includes profiling code. The second object with the profiling code is executed to produce profile information. The profile information is then used when the source code is recompiled in order to produce an optimized object code. As a result, the optimized object code contains executable statements that specifically optimize the code as determined by the profile information. In contrast, the present invention has annotation information (non-executable) in the executable computer program. This allows the executable computer program to be utilized by various analysis tools without generating a unique executable computer program for each type of tool. Thus, Buzbee neither teaches nor suggests reading annotation information *in* an executable computer program (emphasis added).

Applicant has added a new claim, Claim 52, that depends from Claim 19, that includes a further limitation. Claim 52 recites that "the annotation information being generated via an annotation function within source code associated with the executable program." Buzbee clearly does not teach or suggest having an annotation function within the source code. At best, Buzbee teaches to generate profiling code for the second object by dynamically translating the first object code, quite different than having an annotation function within the source code.

Therefore, because dependent Claims 20, 22-23, and 52 include other limitations that are not taught or suggested by Buzbee, for at least the above arguments, Applicant respectfully submits that the § 102(b) rejections of Claims 19, 20 and 22-23 are improper, and respectfully requests reconsideration and withdrawal of these rejections.

Independent Claims 24, 33, 35, 43, and 46, and their respective dependent Claims 25-26, 35, 44-45, and 47-48

The above claims were rejected under 35 U.S.C. § 102(b) as being anticipated by Buzbee. Without unnecessarily repeating the arguments above, Applicant respectfully submits that Claims 24-26, 33, 35, and 43-48 of the present invention are not taught or suggested by Buzbee. For example, Claims 24, 33, and 35 also recite "reading annotation information *in* an executable computer program" (emphasis added). Claim 43 and 45 recite "a receiver of annotation information *from* an executable computer program" (emphasis added. Therefore, as discussed above, Buzbee does not teach or suggest having "non-executable" annotation information "in" or "received from" an executable computer program. Rather, Buzbee teaches to translate the first object code (i.e., an executable) to generate the second object code (i.e., another executable) that has executable profiling code. Buzbee does not even mention having non-executable information within the executable computer program.

Applicant has added an additional dependent claim to each of the above independent claims that adds a further limitation. Each of the new claims, Claims 51-56, recite that "the annotation information being generated via an annotation function within source code associated with the executable program." Thus, as discussed above, Buzbee does not teach or suggest modifying the source code with an annotation function.

In addition, dependent Claims 25-26, 35, 44-45, and 47-48 include other limitations that are not taught or suggested by Buzbee. For at least the above reasons, Applicant respectfully submits that the § 102(b) rejections of Claims 24-26, 33, 35, and 43-48 are improper, and respectfully requests reconsideration and withdrawal of these rejections.

C. 35 U.S.C. § 103 Claim Rejections

Claims 1-11, 13-15, 17, 18, 28-32, 38, 41, and 42 were rejected under 35 U.S.C. § 103(a) as being unpatentable over U.S. Patent No. 5,950,003 to Kaneshiro et al. (hereinafter Kaneshiro) in view of "Compilers: Principles, Techniques, and Tools" by Alfred Aho (hereinafter Aho). Claim 12 was rejected under 35 U.S.C. § 103(a) as being unpatentable over Kaneshiro in view of

Aho and further in view of U.S. Patent No. 5,815,714 to Shridhar (hereinafter Shridhar). Claims 21, 27, 34, and 36 were rejected under 35 U.S.C. § 103(a) as being unpatentable over Buzbee in view of Kaneshiro. Claims 39 and 40 were rejected as being unpatentable over Aho in view of Kaneshiro.

In review, in order to establish a prima facie case of obviousness under 35 U.S.C. § 103, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach or suggest all the claim limitations. The teaching or suggestion to make the claimed combination and the reasonable expectation of success must both be found in the prior art and not be based on Applicant's disclosure. *In re Vaeck*, 947 F.2d 488, 20 USPQ2d 1438 (Fed. Cir. 1991).

A claimed invention is unpatentable as obvious under 35 U.S.C. § 103(a) only if the differences between it and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art. *Ruiz v. A.B. Chance Co.*, 57 USPQ2d 1161, 1165 (Fed. Cir. 2000). In making this determination, the Examiner must carefully avoid the "tempting but forbidden zone of hindsight" in which "that which only the inventor taught is used against its teacher." *In re Dembiczak*, 50 USPQ2d 1614, 1616-1617 (Fed. Cir. 1999). A prima facie case of obviousness based on a combination of references requires a "clear and particular" showing of a teaching or motivation to combine the references. *Winner International Royalty Corp. v. Wang*, 53 USPQ2d 1580, 1587 (Fed. Cir. 2000).

The present invention is directed at analysis tools and methods that read annotation information generated from annotation representations within source code. The analysis tools include debuggers, instruction counters, profilers, fault-injectors, resource failure simulators, optimizers, and the like. Page 1, line 11-12. The tools and methods are adapted to modify executable code generated from source code based on annotation information. For example, if the analysis tool is a debugger, the debugger may insert debugging code into the executable

code. Other tools may be customized to interpret the annotation information in a manner to reflect their desired goal. Thus, several different tools can use the same executable code that has been generated from one source code. The annotation representation within the source code may be an intrinsic function. The annotation information used by the tools remains associated with the code associated with the annotation representation (e.g., the intrinsic function). Page 5, line 16 to Page 6, line 11.

Independent Claim 1 and its dependent Claims 2-13

Claims 1-13 were rejected under 35 U.S.C. § 103(a) as being unpatentable over Kaneshiro in view of Aho.

Kaneshiro is directed at a profile instrumentation method and a profile data collection method. In overview, profiling tools provide a means for obtaining and reporting specific instruction execution times measured by counters or the like, as well as timing information at the procedure, basic block level, and/or instruction level (Col. 1, lines 14-19). Kaneshiro explains that in the past instrumentation codes for identifying the measurement start point and the measurement end point of an appropriate construct of the program was inserted into the original code of the program (Col. 1, lines 59-63). However, Kaneshiro further explains that insertion of this instrumentation code at the beginning of compilation may alter the optimization performed by the compiler. Therefore, the insertion of instrumentation codes must be performed after completion of the code conversions (Col. 1, line 64 to col. 2, line 2).

Kaneshiro teaches a method that is capable of automatically inserting profiling instrumentation codes during compilation. Thus, Kaneshiro teaches to insert instrumentation code into a transformed code of the program during compilation. Kaneshiro also teaches a method for maintaining a mapping between the original code and the transformed code (Col. 3, lines 53-57). For example, in Column 9, Kaneshiro describes two data structures, referred to as "StmtInfo" and "setOfStmtInfo", that are used to maintain the original source code and the transformation history information, respectively.

Aho is a textbook on compiler design. Aho teaches a method for managing symbols using a symbol table. The symbol table is a data structure that contains a record for each identifier used in the source program. Each record has fields for the attributes of the identifier. The symbol table allows the record for each identifier to be found quickly and to store or retrieve data from the record quickly.

Rejected independent Claim 1, from which rejected Claims 2-13 depend, recites "parsing annotation representation *in the source code*" (emphasis added). In the Office Action, the Examiner cited Column 3, lines 51-55, of Kaneshiro for teaching this limitation. However, upon a closer inspection of Column 3, lines 51-55, the Applicant submits that Kaneshiro does not teach this limitation. Rather, as explained above, Kaneshiro provides a "profile instrumentation method capable of *automatically inserting profiling instrumentation codes during compilation*" (emphasis added). In fact, in the Background Section in Column 1, lines 64 to column 2, line 1, Kaneshiro explains that inserting instrumentation codes at the beginning of compilation (e.g., in the source code) creates a possibility that the instrumentation will slightly alter the optimization performed by the compiler. Thus, in fact, Kaneshiro teaches that the source code should not be modified and teaches away from having "annotation representation in the source code" as recited in Claim 1.

Rejected independent Claim 1 also recites "transforming the annotation representation into intermediate language code". The Examiner stated that "since the program code contains annotations within the code, when the code is compiled (Figure 11, item S21), it will be transformed into intermediate language code. The Applicant has thoroughly reviewed Kaneshiro and is unable to determine the origin of this proposition. There is no mention in Kaneshiro that the source code contains annotations. Rather, as described above, Kaneshiro, teaches to automatically insert instrumentation codes into the transformed code. In fact, Kaneshiro teaches away from making modification to the source code.

Rejected independent Claim 1 also recites "generating annotation information from the intermediate language code." The Examiner concedes that Kaneshiro does not teach generating a non-executable statement from the intermediate language code. However, the Examiner

contends that Aho describes a symbol table that the compiler creates and uses to keep track of variables and functions in the source code. Therefore, the Examiner contends that it would have been obvious to one of ordinary skill in the art to pass annotation representation into the source code, transform the representation into intermediate language code, where a symbol is generated for the annotation representation. The Applicant respectfully disagrees. First, Aho creates a symbol table from the source code. Thus, Aho does not teach generating annotation information from *the intermediate language code*. (emphasis added). Secondly, there is no motivation to combine these disparate references. For at least the above arguments, Applicant respectfully submits Claim 1 of the present invention is not obvious in view of Kaneshiro, whether considered alone or with any permissible combination with the prior art of record, including Aho.

In addition, dependent Claims 2-13 include other limitations that are not taught or suggested by Kaneshiro, whether considered alone or with any permissible combination with the prior art of record. For at least the above reasons, Applicant respectfully submits that the 103(a) rejections of Claims 1-13 are improper, and respectfully request reconsideration and withdrawal of these rejections.

Independent Claim 14 and its dependent Claims 15, 17, and 18

Claims 14, 15, 17, and 18 were rejected under 35 U.S.C. § 103(a) as being unpatentable over Kaneshiro in view of Aho.

Rejected independent Claim 14, from which rejected Claims 15, 17 and 18 depend, recites "*annotating computer source code*" (emphasis added). In the Office Action, the Examiner cited Column 5, lines 65-67, of Kaneshiro for teaching this limitation. Column 5, lines 65-67, state that "the codes to be profiled have already been instrumented using calls to profile library subroutines." However, elsewhere in Kaneshiro, Kaneshiro describes these profile run-time library subroutines. Specifically, arguments for these profile run-time library subroutines are associated with descriptive strings conveying information about the original source code. The strings must contain enough information to map feed back profile information to the original

source code and to be readable by a user when being accessed by a profile data display tool. Col. 8, lines 57-63. Thus, the Applicant contends that Kaneshiro does not teach or suggest "annotating computer source code" as recited in Claim 14. In fact as discussed above, Kaneshiro provides a "profile instrumentation method capable of *automatically inserting profiling instrumentation codes during compilation*" (emphasis added). Thus, these codes are automatically inserted into the executable during compilation. In contrast, Claim 14 recites "annotating computer source code." In fact, in the Background Section in Column 1, line 64 to column 2, line 1, Kaneshiro explains that inserting instrumentation codes at the beginning of compilation (e.g., in the source code) creates a possibility that the instrumentation will slightly alter the optimization performed by the compiler. Thus, in fact, Kaneshiro teaches away from modifying the source code and "annotating computer source code" as recited in Claim 14.

In regards to Claims 15-18, the Examiner concedes that Kaneshiro does not teach generating a symbol having string parameters of the function call and emitting the symbol to a symbol table. However, the Examiner contends that Aho describes a symbol table that the compiler creates and uses to keep track of variables and functions in the source code. The symbol table keeps tack of a variety of parameters of a procedure. Thus, the Examiner contends that it would have been obvious to one of ordinary skill in the art to annotate computer source code with an intrinsic function call, generate annotation information from this function call, and store this information in an output file where a symbol having string parameters is created for the function and stored on a symbol table. The Applicant respectfully disagrees.

First, Kaneshiro fails to teach or suggest the limitations as described above. Second, Aho merely describes a data structure for storing information about identifiers used in a source program. Aho further teaches that for a procedure name, the attributes may include the number and types of the arguments, a method for passing each argument and the type returned, if any. Page 11, lines 5-7. Thus, "generating a symbol having string parameters of the intrinsic function call" is not taught or suggested by Aho. Third, there is no motivation or suggestion to combine Aho and Kaneshiro. Thus, the Applicant contends that Kaneshiro, whether considered alone or with any permissible combination with the prior art of record, including Aho, does not teach or

suggest "annotating computer source code" as recited in Claim 14. In addition, dependent Claims 15-18 include other limitations that are not taught or suggested by Kaneshiro, whether considered alone or with any permissible combination with the prior art of record. For at least the above reasons, Applicant respectfully submits that the 103(a) rejections of Claims 14, 15, 17, and 18 are improper, and respectfully requests reconsideration and withdrawal of these rejections.

Independent Claim 28 and its dependent Claims 29-32

Claims 28-32 were rejected under 35 U.S.C. § 103(a) as being unpatentable over Kaneshiro in view of Aho.

The Examiner rejected Claim 28 for the same reasons as Claims 14 and 15. The Examiner contends that Kaneshiro teaches annotating computer source code using intrinsic function calls in the source code as recited in Claim 14. Therefore, without unnecessarily repeating the arguments above, Applicant respectfully submits that Claims 28-32 of the present invention are not taught or suggested by Kaneshiro, whether considered alone or with any permissible combination with the prior art of record, including Aho. Each claim now recites "parsing an intrinsic annotation function call within source code." Thus, as discussed above, the cited prior art teaches away from modifying the source code or adding code to the source code. In addition, dependent Claims 29-32 each recite other limitations that are not taught or suggested by the prior art. For at least the above reasons, Applicant respectfully submits that the 103(a) rejections of Claims 28-32 are improper, and respectfully requests reconsideration and withdrawal of these rejections.

Dependent Claims 21, 27, 34 and 36

Claims 21, 27, 34, and 36 were rejected under 35 U.S.C. § 103(a) as being unpatentable over Buzbee in view of Kaneshiro.

The Examiner concedes that Buzbee fails to teach that "the annotation information is generated by a function call having at least one string parameter" as recited in Claim 21.

However, the Examiner contends that Kaneshiro taught attaining annotation information by means of a function call with at least one string parameter. Therefore, the Examiner contends that it would have been obvious to one of ordinary skill in the art to control the execution of a computer analysis tool using annotation information as taught by Buzbee, where the annotation information is generated by a function call containing parameters, since function calls are much easier and much more compact, and hence a neater way of running multiple lines of code in a source program.

The Applicant respectfully disagrees. First, as discussed above, Buzbee clearly does not teach or suggest having an "intrinsic annotation function call within the source code" as recited Claims 21, 27, 34 and 36. At best, Buzbee teaches to generate profiling code for the second object by dynamically translating the first object code, quite different than having an annotation function within the source code. In addition, in contrast to Examiner's contention that Kaneshiro teaches to attain annotation information by means of a function call, Applicant contends that Kaneshiro teaches away from modifying the source code. Thus, Kaneshiro does not teach attaining annotation information from an intrinsic annotation function call *that the executable computer program was compiled from* (emphasis added). Third, there is no motivation to combine these references.

Thus, Applicant respectfully submits that Claims 21, 27, 34 and 36 of the present invention are not taught or suggested by Buzbee, whether considered alone or with any permissible combination with the prior art of record, including Kaneshiro. For at least the above reasons, Applicant respectfully submits that the 103(a) rejections of Claim 34 and 36 are improper, and respectfully requests reconsideration and withdrawal of these rejections.

Independent Claim 38

Claim 38 was rejected under 35 U.S.C. § 103(a) as being unpatentable over Kaneshiro in view of Aho.

Applicant respectfully submits that Claim 38 of the present invention is not taught or suggested by Kaneshiro, whether considered alone or with any permissible combination with the

prior art of record, including Aho. Claim 38 now recites "parsing a source annotation representation within source code." As discussed above, Kaneshiro teaches away from modifying the source code. For at least the above reasons, Applicant respectfully submits that the 103(a) rejection of Claim 38 is improper, and respectfully requests reconsideration and withdrawal of this rejection.

Independent Claim 39 and its dependent Claim 40

Claims 39 and 40 were rejected under 35 U.S.C. § 103(a) as being unpatentable over Aho in view of Kaneshiro.

In regards to Claim 39, the Examiner contends that Aho teaches a data structure that contains information corresponding to a function in a computer program. Aho describes a symbol table that the compiler creates and uses to keep track of variables and functions in the source code. The symbol table keeps track of a variety of arguments of a procedure, as well as the procedure name. However, the Examiner concedes that Aho does not teach that the functions are annotation functions, and that the information is annotation information. However, the Examiner contends that Kaneshiro does teach using annotation functions to collect annotation information. Therefore, the Examiner contends that it would have been obvious to one of ordinary skill in the art to store a data structure comprising information corresponding to a function in a source code as taught by Aho, where the function is an annotation function, as taught by Kaneshiro, since this allows the compiler to keep track of names generated in the computer program.

The Applicant agrees with the Examiner that Aho does not teach that the functions are annotation functions. However, the Applicant respectfully disagrees that Kaneshiro teaches this limitation. As discussed above, Kaneshiro teaches away from modifying the source code. Thus, Kaneshiro fails to teach or suggest an annotation function in a source computer program as recited in the claims. Thus, Applicant respectfully submits that Claims 39 and 40 of the present invention are not taught or suggested by Aho, whether considered alone or with any permissible combination with the prior art of record, including Kaneshiro. For at least the above reasons,

Applicant respectfully submits that the 103(a) rejections of Claims 39 and 40 are improper, and respectfully requests reconsideration and withdrawal of these rejections.

Independent Claim 41 and its dependent Claim 42

Claims 41 and 42 were rejected under 35 U.S.C. § 103(a) as being unpatentable over Kaneshiro in view of Aho.

The Examiner stated that Claim 41 corresponded directly with Claim 28 and was rejected for the same reason as Claim 28. In review, the Examiner rejected Claim 28 for the same reasons as Claims 14 and 15. Therefore, without unnecessarily repeating the arguments above, Applicant respectfully submits that Claims 41 and 42 of the present invention are not taught or suggested by Kaneshiro, whether considered alone or with any permissible combination with the prior art of record, including Aho. Claim 41 recites "an annotation function call in source code". As discussed above, Kaneshiro teaches away from having an annotation function within the source code or modifying the source code at all. For at least the above reasons, Applicant respectfully submits that the 103(a) rejections of Claims 41 and 42 are improper, and respectfully requests reconsideration and withdrawal of these rejections.

III. CONCLUSION

Applicant has considered the other references cited by the Examiner in the Office Action. None of these references appear to affect the patentability of Applicant's claims. By the foregoing amendments and remarks, Applicant believes that all pending claims are allowable and the application is in condition for allowance. Therefore, a Notice of Allowance is respectfully requested. Should the Examiner have any further issues regarding this application, the Examiner is requested to contact the undersigned attorney for the Applicant at the telephone number provided below.

Respectfully submitted,

MERCHANT & GOULD P.C.

Marcia A. Tunheim
Registration No. 42,189
Direct Dial: 206.342.6259

MERCHANT & GOULD P.C.
P. O. Box 2903
Minneapolis, Minnesota 55402-0903
206.342.6200